(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 676 699 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **95302002.1**

(51) Int. Cl.[6] : **G06F 13/18**

(22) Date of filing : **24.03.95**

(30) Priority : **04.04.94 US 223405**

(43) Date of publication of application :
11.10.95 Bulletin 95/41

(84) Designated Contracting States :
DE FR GB

(71) Applicant : **SYMBIOS LOGIC INC.**
2001 Danfield Court
Fort Collins, Colorado 80525 (US)

(72) Inventor : **Dekoning, Rodney A.**
6443 Danbury
Wichita, KS67226 (US)
Inventor : **Hoglund, Timothy E.**
3035 Purgatory Drive
Colorado Springs, CO 80918 (US)

(74) Representative : **Gill, David Alan**
W.H. Beck, Greener & Co.,
7 Stone Buildings,
Lincoln's Inn
London WC2A 3SZ (GB)

(54) Method of managing resources shared by multiple processing units.

(57) The invention provides for resource allocation logic for a computer system including a plurality of processors (17, 21) which share access to, and control of, a plurality of resources, such as disk drive units (31-35, 41-45) or busses (51-55). Resource allocation logic (300) coordinates the execution of requests received from the processors to avoid resource sharing inefficiencies and deadlock situations. The allocation logic (300) maintains a "request" queue for each processor (17, 21), seeking to satisfy all requests quickly and fairly. The queues contain an entry corresponding to each request received from its corresponding processor and an identification of resources that are required by the entry's corresponding request. The allocation logic (300) also maintains a "resources available" status array of resources which are not currently in use by any processors, or are not reserved for future use by any processors. The logic repeatedly compares each entry in the request queues with the entries in the resources available status array to detect an entry in the request queue identifying resources all of which are contained in the resources available status array. Once the allocation logic (300) can satisfy a particular request, it signals a grant to the requesting processor for the resources requested and the requested resources are removed from the resources available status array. Upon conclusion of execution of the granted request, the resources are again released to the resource allocation logic (300) for utilization by other resource requests.

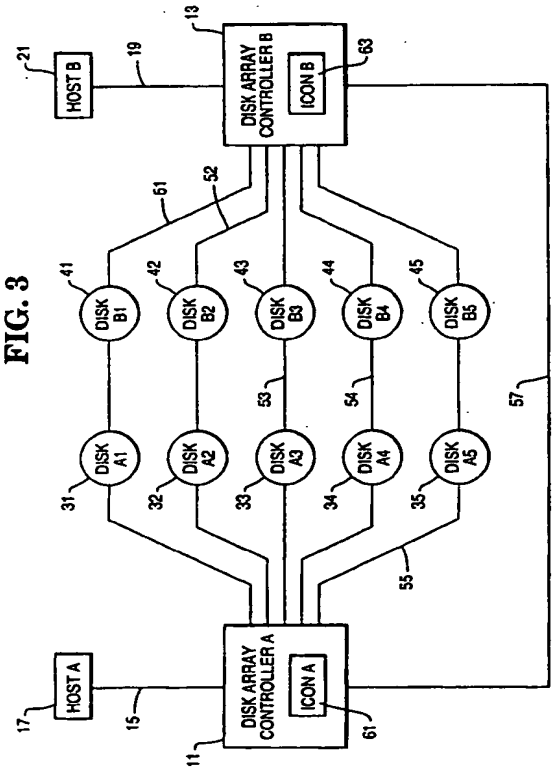


FIG. 3

EP 0 676 699 A2

RNSDOCID: <EP 0676699A2 I >

The present invention relates to a method of managing resources shared by multiple processing units.

In particular, the present invention can provide for a method of managing the operations of multiple disk array controllers which share access to the disk drive units within the array.

Disk array storage devices comprising a multiplicity of small inexpensiv disk drives, such as the 5 ¼ or 3 ½ inch disk drives currently used in personal computers and workstations, connected in parallel are finding increased usage for non-volatile storage of information within computer systems. The disk array appears as a single large fast disk to the host system but offers improvements in performance, reliability, power consumption and scalability over a single large magnetic disk.

Most popular RAID (Redundant Array of Inexpensive Disks) disk array storage systems include several drives for the storage of data and an additional disk drive for the storage of parity information. Thus, should one of the data or parity drives fail, the lost data or parity can be reconstructed. In order to coordinate the operation of the multitude of drives to perform read and write functions, parity generation and checking, and data restoration and reconstruction, many RAID disk array storage systems include a dedicated hardware controller, thereby relieving the host system from the burdens of managing array operations. An additional or redundant disk array controller (RDAC) can be provided to reduce the possibility of loss of access to data due to a controller failure.

The present invention seeks to provide for advantages having regard to the management of shared resources such as disk arrays.

In accordance with one aspect of the present invention, there is provided a method of coordinating the execution of I/O requests received from requesting agents in a computer system in which said requesting agents share access to and control over a plurality of resources, characterized by the steps of:

(A) establishing a request queue which includes an entry corresponding to each I/O request received from said requesting agents, each entry including an identification of resources that are required by said entry's corresponding I/O request;

(B) maintaining a resources available status array which includes an entry for each resource which is not currently in use by any requesting agent and is not currently reserved for future use by any requesting agent;

(C) systematically comparing each entry in said request queue with the entries in said resources available status array to detect an entry in said request queue identifying resources all of which are contained in said resources available status array;

(D) granting control of the resources associated with said entry detected in step C to the requesting agent providing the I/O request corresponding to the entry detected in step C; and

(E) executing the I/O request corresponding to the entry identified in step C.

In accordance with another aspect of the present invention there is provided a method of coordinating the operation of processors, in a disk array system including first and second disk array controllers and a plurality of disk drives and busses under the control of each of said processors, characterized by the steps of:

(A) establishing a request queue for each disk array controller which includes an entry corresponding to an I/O request received by said disk array system, each entry including an identification of disk drives and busses that are required by said entry's corresponding I/O request;

(B) maintaining a resources available status array which includes an entry for each disk drive and bus which is not currently in use by any disk array controller and is not currently reserved for future use by any disk array controller;

(C) systematically comparing each entry in said request queues with the entries in said resources available status array to detect an entry in said request queues identifying resources all of which are contained in said resources available status array;

(D) granting control of the disk drives and busses associated with said entry detected in step C to the disk array controller associated with the request queue corresponding to the entry detected in step C; and

(E) executing the I/O request corresponding to the entry identified in step C.

The present invention, is particularly advantageous in providing a new and useful method and structure for coordinating the operation of multiple controllers which share access to and control over common resources.

In particular the present invention advantageously can provide such a method and structure which reduces or eliminates resource sharing inefficiencies and deadlock situations which arise in systems which include shared resources.

Preferably, the present invention can also provide a new and useful disk array storag system including multiple active array controllers.

Further, the present invention can provide a method for coordinating the operation of multiple active controllers within a disk array which share access to and control over common resources.

VSDOCID: <EP___0676699A2_I_>

According to one particular feature, the present invention provides a new and useful method for avoiding contention between controllers in a disk array system including multiple active controllers.

In particular from the above, it will be appreciated that the present invention can comprise a method for coordinating the execution of requests received from multiple requesting agents which share access to and control over common resources within a computer system in order to avoid resource sharing inefficiencies and deadlock situations. The method includes the steps of: (A) establishing a "request" queue, said request queue including an entry corresponding to each request received from the requesting agents, each entry including an identification of resources that are required by said entry's corresponding request; (B) maintaining a "resources available" status array, said resources available status array including an entry for each resource which is not currently in use by any requesting agent and is not currently reserved for future use by any requesting agent; (C) systematically comparing each entry in said request queue with the entries in said resources available status array to detect an entry in said request queue identifying resources all of which are contained in said resources available status array; (D) granting control of the resources associated with said entry detected in step C to the requesting agent providing the request corresponding to the entry detected in step C; and (E) executing the request corresponding to the entry identified in step C. The resources associated with the granted request are removed from the resources available status array during the execution of step (E). Upon conclusion of execution of the granted request, the resources are again placed in the resources available status array for utilization by other resource requests.

A particular embodiment may be incorporated into a disk array subsystem including multiple array controllers which share access to, and control over, multiple disk drives and control, address and data busses within the disk array. A request queue containing entries for I/O requests received from the host computer system is maintained for each array controller, the method of the present invention alternately examining entries in each request queue to detect an entry in either request queue identifying resources all of which are contained in the resources available status array. Additionally, each request queue contains a list age indicating the relative age of each request queue with respect to the other request queues, and each entry in the request queues includes a request age indicating the relative age of each entry in a request queue with respect to other entries in the request queue. In examining the request queues to identify I/O requests for execution, priority is awarded to entries based on the relative ages of the request queues and request queue entries.

The invention is described further hereinafter, by way of example only, with reference to the accompanying drawings in which :

Fig. 1 is a block diagram representation of a disk array system including two SCSI host busses, dual disk array controllers; and ten disk drives accessed through five SCSI busses shared by the dual controllers;

Fig. 2 is a block diagram representation of a disk array system including dual disk array controllers connected to a common SCSI host bus, and ten disk drives accessed through five SCSI busses shared by the dual controllers;

Fig. 3 is a block diagram representation of a disk array system including dual active controllers and a communication link between the controllers for providing communications and coordinating resource arbitration and allocation between the dual active disk array controllers; and

Fig. 4 is a block diagram, comprising Figs. 4A, 4B and 4C, of the ICON (Inter-Controller Communication Chip) ASIC (Application Specific Integrated Circuit) incorporated into each disk array controller of Figure 3 for providing communications and coordinating resource arbitration and allocation between the dual active disk array controllers.

Figure 1 is a block diagram representation of a disk array storage system including dual disk array controllers 11 and 13. Array controller 11 is connected through a SCSI host bus 15 to host system 17. Array controller 13 is likewise connected through a SCSI host bus 19 to a host system 21. Host systems 17 and 21 may be different processors in a multiple processor computer system. Each array controller has access to ten disk drives, identified by reference numerals 31 through 35 and 41 through 45, via five SCSI busses 51 through 55. Two disk drives reside on each one of busses 51 through 55. Disk array controllers 11 and 13 may operate in one of the following arrangements:

(1) Active/Passive RDAC.

All array operations are controlled by one array controller, designated the active controller. The second, or passive, controller is provided as a hot spare, assuming array operations upon a failure of the first controller.

(2) Active/Active RDAC -- Non Concurrent Access of Array Drives. On controller has primary responsibility for a first group of shared resources (disk drives, shared busses), and stand-by responsibility for a second group of resources. The second controller has primary responsibility for the second group of resources and stand-by responsibility for the first group of resources. For xample, disk array controller 11 may have primary responsibility for disk drives 31 through 35, while disk array controller has primary re-

ENSDOCID: <EP__0676699A2_I_>

sponsibility for disk drives 41 through 45.

(3) Active/Active RDAC – Concurr nt Access of Array Drives. Each array controller has equal access to and control over all resources within the array.

Providing each array controller with equal access to, and control over, shared resources may lead to resource sharing inefficiencies or deadlock scenarios. For example, certain modes of operation require that subgroups of the channel resources be owned by one of the array controllers. Failure to possess all required resources concurrently leads to blockage of the controller until all resources have been acquired. In a multiple controller environment obtaining some but not all the required resources for a given transaction may lead to resource inefficiencies or deadlock in shared resource acquisition.

Likewise, an array controller that provides hardware assistance in generating data redundancy requires simultaneous data transfer from more than one drive at a time. As data is received from the drives or the host, it is passed through a RAID-striping ASIC to generate data redundancy information that is either stored in controller buffers or passed immediately to a drive for storage. Each controller must have access to multiple selected drive channels concurrently so that the data may be passed through the RAID striping ASIC from the multiple data sources concurrently. Deadlock can occur if no means to coordinate access to the drive channels exists.

Two examples are given below to illustrate the deadlock situation in a two disk array controller environment.

Deadlock Condition 1:

Referring to Figure 1, disk array controllers 11 and 13 are seen to share five SCSI buses 51 through 55 and the ten drives that are connected to the SCSI buses. Disk array controller 11 is requested to perform an I/O operation to transfer data from drives disk drive 31 and 33. Simultaneously, disk array controller 13 is requested to perform an I/O operation to transfer data from disk drives 41 and 43. Both disk controllers attempt to access the drives they need concurrently as follows:

Array controller 11 acquires bus 51 and disk drive 31, and is blocked from acquiring bus 53 and disk drive 33.

Array controller 13 acquires bus 53 and disk drive 43, and continues arbitrating for bus 51 and disk drive 41.

Controller 11 now has SCSI bus 51 in use, and is waiting for disk drive 33 on SCSI bus 53 (owned by Controller 13). Controller 13 now has SCSI bus 53 in use, and is waiting for disk drive 41 on SCSI bus 51 (owned by Controller 1).

Deadlock Condition 2:

Deadlock can occur when multiple controllers are attached to the same host bus. This may occur when host SCSI bus 15 and host SCSI bus 19 are the same physical SCSI bus, identified as bus 27 in Figure 2. Controller 11 is requested to perform an I/O operation requiring a transfer of data from disk drive 31 on SCSI bus 51 to host 17. Simultaneously, controller 13 is requested to perform an I/O operation requiring a transfer of data from disk drive 41 on SCSI bus 51 to host 21. Both controllers attempt access of the resources they need concurrently as follows:

Array controller 11 acquires the single Host SCSI bus, identified by reference numeral 27 and is blocked from acquiring SCSI bus 51 and disk drive 31.

Array controller 13 acquires SCSI bus 51 and disk drive 41, and is blocked from acquiring the host SCSI bus 15.

Controller 11 now has the host SCSI bus 27 in use, and is waiting for access to SCSI bus 51 (owned by Controller 13.) so that it can connect to disk drive 31. Controller 13 now has SCSI bus 51 in use, and is waiting for access to the host SCSI bus 27 (owned by Controller 1.).

A method and structure for coordinating the operation of multiple controllers which share access to and control over common resources is required to eliminate resource sharing inefficiencies and deadlock situations.

A disk array system including dual active controllers constructed in accordance with a preferred embodiment f the pr sent invention is shown in block diagram form in Figure 3. In addition to the structure shown in the disk array system of Figure 1, the system of Figure 3 includes a dedicated communication link 57 connected between the array controllers 11 and 13, and an ICON-ASIC incorporated into each of the controll rs, identified by reference numerals 61 and 63, respectively.

The communication link 57 and ICON chip provide communication between, and resource arbitration and

NSDOCID: <EP___0676699A2_I_>

allocation for the dual disk array controllers.

Figure 4 is a block diagram of the ICON chip incorporated into each of the dual active array controllers 11 and 13 included within the disk array system shown in Figure 3. The ICON chip contains all functions necessary to provide high speed serial communication and resource arbitration/allocation betw en two Disk Array controllers. The primary application for the ICON chip is in Disk Array systems utilizing redundant disk array controllers. Because the redundant controller configuration shares resources (disk drives and SCSI buses) between two controllers, a method of arbitrating for these common resources must be utilized in order to prevent deadlocks and to maximize system performance. The ICON chip contains a hardware implementation of a resource allocation algorithm which will prevent deadlocks and which strives to maximize system performance. In addition to performing resource arbitration/allocation, the ICON chip also provides a means of sending/receiving generic multiple byte messages between Disk Array controllers. The ICON chip includes the following logic modules:

Microprocessor Interface Control Logic 100.

The microprocessor interface block 100 allows an external microprocessor to configure and monitor the state of the ICON chip. Configuration and status information are maintained in registers within the ICON chip. The configuration, control, and status registers are designed to provide operating software with a wide range of functionality and diagnostic operations. Interrupt masking and control are also included in this functional block.

Inter-controller Communication Logic 200.

The Inter-controller Communication block 200 contains all structures and logic required to implement the inter-controller communication interface. This block includes the following structures/logic: Send State Sequencer 201, Receive State Sequencer 203, Message Send Buffer 205, Message Receive Buffer 207, Status Send Register 209, and Status Receive Buffer 211. These modules work together to form two independent unidirectional communication channels. Serialization and Deserialization of data packets occurs in Send State Sequencer 201 and Receive State Sequencer 203 modules. Serial data output from the Send State Sequencer may be fed into the Receiver State Sequencer module for a full diagnostic data turnaround.

The Inter-controller Communication Block is used to send generic messages and status or to send specific request/grant/release resource messages between two Disk Array controllers.
Communication between pairs of ICON chips is provided by six signals. These signals are defined as follows:

Table 1

Communication Signal Descriptions		
Name	**Type**	**Description**
ARDY/	OUT	'A' Port ready. This output is controlled by the ICON Ready bit in the Control Register and is monitored by the alternate controller.
BRDY/	IN	'B' Port ready. This input is used to monitor the Ready/Not Ready status of the alternate controller.
AREQ.DAT/	OUT	'A' Port Requuest/Serial Data. This output signal is used to request data transfer and then send serial data to the alternate controller in response to the 'A' Port Acknowledge signal.
BREQ.DAT/	IN	'B' Port Request/Serial Data. This input is used to receive serial data from the alternate controller.
AACK/	IN	'A' Port Acknowledge. This signal is received from the alternate controller as the handshake for a single data bit transfer.
BACK/	OUT	'B' Port Acknowledge. This output signal is sent to the alternat controller to control a serial receiv data transfer operation.

ENSDOCID: <EP 0676699A2 I >

Resource Allocation Logic 300.

The Resource Allocation block 300 contains all structures and logic required to manage up to ight shared resources between two Disk Array controllers, referred to as the master and slave disk array controllers. These structures/logic include the Resource Allocator 301, two sets of Resource Request Lists (Master/Slave) 303 and 305, two sets of Release Resource FIFOs (Master/Slave) 307 and 309, two sets of Resources Granted FIFOs (Master/Slave) 311 and 313, and the Resource Scoreboard comprising resources allocated and resources available blocks 315 and 317, respectively.

The key element in this block is Resource Allocator 301. This block consists of a hardware implementation of an intelligent resource allocation algorithm. All other data structures in this block are directly controlled and monitored by the Resource Allocator. The Resource Allocator present in the ICON chip for the master controller continually monitors the state of the Resource Request Lists, the Release Resource FIFOs, and Resource Scoreboard to determine how and when to allocate resources to either controller. The Resource Allocator present in the ICON chip for the slave controller is not active except during diagnostic testing.

Controller Functions 400.

The Controller Functions logic 400 provides several board-level logic functions in order to increase the level of integration present on the disk array controller design.

The invention advantageously encompasses the establishment of a simple communication link and protocol between devices sharing resources, and a unique arbitration algorithm which is used for the management of the shared resources.

The communication link and protocol are used to request, grant, and release resources to or from the resource arbiter. The protocol requires the establishment among the devices sharing resources of a single master device, and one or more slave devices. The master/slave distinction is used only for the purposes of locating the active resource allocation logic 300. Although each controller includes resource allocation logic, this logic is only active in the master controller. In the discussion which follows, references to the resource allocation logic 300 and its components will refer to the active resource allocation logic and its components. Both master and slave devices retain their peer to peer relationship for system operations.

The active resource allocator 301 is implemented in the master device. A device formulates a resource request by compiling a list of resources that are required for a given operation. The resource request is then passed to the resource allocation logic 300. The resource allocation logic 300 maintains a list of requests for each device in the system, seeking to satisfy all requests quickly and fairly. Once the allocation logic can satisfy a particular request, it signals a grant to the requesting device for the resources requested. The device with the granted resource requests has access to the granted resources until it releases them. The release is then performed by sending a release message to the resource allocator to free the resources for consumption by other resource requests.

All resource requests, request granting, and request freeing involving a slave device is performed by sending inter-device messages, which include message type and data fields, between the master (where the active resource allocation logic is located) and the slave devices using the interface described above. All resource requests, request grants, and request freeing involving only the master device may be done within the local to the master device.

The resource allocation logic 300 located in the arbitrarily assigned master device includes a resource allocation algorithm and associated data structures for the management of an arbitrary number of shared resources between an arbitrary number of devices. The data structures and algorithm for sharing resources are discussed below.

Data Structures.

For each device which requires shared resource management, a request queue, or list of resource requests, of arbitrary depth is maintained by the master device (master and slave request lists 303 and 305). Associated with each of the device request queues are two count values, a list age (which indicates the relative age of a device request queue with respect to the other request queues) and a request age (which indicates the relative age of the oldest entry in a single device's request queue with respect to other entries in the same request queue). In addition to the count values associated with each device request queue, two bool an flags are also maintained; a Request Stagnation flag and a List Stagnation flag. Request Stagnation TRUE indicates that the relative age of a device's oldest resource request has exceeded a programmable threshold value. List Stagnation TRUE indicates that the relative age of a device's request queue with respect to other devices' re-

SDOCID: <EP___0676699A2_I_>

quest queues has xceeded a programmable threshold value. Stagnation (Request or List) is mutually exclusive between all devices, only one device can be in the Stagnant state at any given time.

The master device also maintains th current state of resource allocation and reservation by tracking "Resources Available" and "Resources Reserved". "Resources Available" indicates to the r source allocation algorithm which resources are not currently in use by any device and are not currently reserved for future allocation. Any resources contained within the "Resources Available" structure (Resources Available block 317) are therefore available for allocation. "Resources Reserved" indicates to the resource allocation algorithm which resources have been reserved for future allocation due to one of the devices having entered the Stagnant state (Request Stagnation or List Stagnation TRUE). Once a device enters the Stagnant state, resources included in the stagnant request are placed into the "Reserved Resources" structure (Resource Reserved block 315) either by immediate removal from the "Resources Available" structure, or for resources currently allocated, at the time they are released or returned to the resource pool) and kept there until all resources included in the stagnant request are available for granting. Stagnation (Request or List) is mutually exclusive between all devices; only one device can be in the Stagnant state at any given time. The last two data structures used by the resource allocation algorithm are pointers to the currently selected device (generically termed TURN and LISTSELECT) which is having it's resource request queue being searched for a match with available resources.

Algorithm.

Resource allocation fairness is provided using the above-defined data structures. The Request Stagnation flag as previously described is used to ensure fairness in granting resource requests within a single device. For example, assuming random availability of resources, a device which requests most resources in groupings of two could starve it's own requests for groupings of five resources from the same resource pool unless a mechanism for detecting and correcting this situation exists. The request age counts with their associated thresholds ensure that resource requests within a single device will not be starved or indefinitely blocked.

The List Stagnation flag is used to ensure fairness in granting resource requests between devices. For example, a device which requests resources in groupings of two could starve another device in the system requesting groupings of five resources from the same resource pool. The list age counts with their associated thresholds ensure that all devices' requests will be serviced more fairly and that a particular device will not become starved waiting for resource requests.

Two modes of operation are defined for the resource allocation algorithm: Normal mode and Stagnant mode. Under Normal mode of operation, no devices have entered the Stagnant state and the algorithm uses the TURN pointer in a round-robin manner to systematically examine each of the device's request queues seeking to grant any resources which it can (based on resource availability) with priority within a device request queue based on the relative ages of the request entries. Upon transition to the Stagnant mode (a device has enter the Stagnant state), the TURN pointer is set to the Stagnant device and the resource allocation algorithm will favor granting of the request which caused the Stagnant state by reserving the resources included in the stagnant request such that no other device may be granted those resources. Although the TURN pointer is effectively frozen to the Stagnant device, other device request queues and other entries within the Stagnant device's request queue will continue to search for resource matches based on what is currently available and not reserved using the secondary list pointer (LISTSELECT).

The actual resource grant operation includes the removal of granted resources from the "Resources Available" structure along with the clearing of "Resources Reserved" structure (if the resource grant was for a Stagnant request). Resource freeing or release operations are accomplished simply by updating the "Resources Available" structure.

A Specific Resource Algorithm implementation.

The following is an implementation of the algorithm using the "C" programming language for a sample case of a master and a single slave device with the following characteristics:

- Resource Request Queue depth for both devices = 4
- Number of Shared R sources b tween the devices = 8

As stated earlier, the number of devices, number of shared resources, and queue depth are strictly arbitrary. The functionality contained and implied by this algorithm is implemented in the device sharing the resources designated the master. The description describes the service poll used to look for a resource request to be granted from any controller. The release operation is simply provided by allocating th resources to be released to the channels available variable.

NSDOCID: <EP 0676699A2 I >

Although this xample implementation uses the "C" programming language, the implementation may take any form, such as other programming languages, hardware state machine implementations, etc.

```
void resource_allocation_algorithm(void)                    /* begin resource allocation
algorithm */
{
int service_loops;
resource_operation *stagnant_operation;

/* while ((q_head_is_not_empty(slave_list)) && (q_head_is_not_empty(master_list)))
*/
for (service_loops = 0; service_loops < 4; service_loops++)
    {

    if (service_loops == 0)
        {
        if ((!master_request_stagnation) && (!master_list_stagnation) &&
            (!slave_request_stagnation) && (!!slave_list_stagnation))
            {
            if (last_serviced == MASTER)
                turn = SLAVE;
            else
                turn = MASTER;
            }
        }
```


NSDOCID: <EP___0676699A2_I_>

```
if (turn == MASTER)
    {
    if (!(q_head_is_not_empty(master_list)))
        {
        master_list_age = 0;
        turn = SLAVE;
        continue;
        }

    if ((!master_list_stagnation) && (!master_request_stagnation))
        {
        if (acquire_from_master())
            {
            if (oldest_master_serviced)
                {
                turn = SLAVE;
                master_request_age = 0;
                }
            else
                {


                master_request_age++;
                if (master_request_age >= request_threshold)
                    {


                    num_master_req_stagnation++;
```


XSDOCID: <EP 0676699A2 I >

```
                    master_request_stagnation = TRUE;
                    }
                else
                    {
                    turn = SLAVE;
                    }
                }
            /* a request from the master queue was serviced */
            master_list_age = 0;
            slave_list_age++;
            if ((slave_list_age >=
list_threshold)&&(!master_request_stagnation))
                {
                if (q_head_is_not_empty(slave_list))
                    {
                    num_slave_list_stagnation++;

                    slave_list_stagnation = TRUE;
                    turn = SLAVE;
                    }
                }
            }
        else
            {
            /* no master queue request was serviced */
            turn = SLAVE;
```


ISDOCID: <EP___0676699A2_I_>

```
            }
        }
    else
        {
        /* master_list_stagnation or master_request_stagnation */
        stagnant_operation = (resource_operation*)master_list->head;
        slave_list_age = 0;
        if (acquire_from_master())
            {
            if (oldest_master_serviced)
                {

                turn = SLAVE;
                master_list_stagnation = FALSE;
                master_request_stagnation = FALSE;
                slave_list_age++;
                master_request_age = 0;
                master_list_age = 0;
                }
            }
        else
            {
            if (acquire_from_slave())
                {
                slave_list_age = 0;
                if (oldest_slave_serviced)
                    slave_request_age = 0;
```


BNSDOCID: <EP 0676699A2 I >

```
                    }
                }
            }
        }
    else
        {
        /* turn = SLAVE */
        if (!(q_head_is_not_empty(slave_list)))
            {
            slave_list_age = 0;
            turn = MASTER;
            continue;
            }

        if ((!slave_list_stagnation) && (!slave_request_stagnation))
            {
            if (acquire_from_slave())
                {
                if (oldest_slave_serviced)
                    {
                    turn = MASTER;
                    slave_request_age = 0;
                    }
                else
                    {
```


ISDOCID: <EP___0676699A2_I_>

```
slave_request_age++;
if (slave_request_age >= request_threshold)
    {

    num_slave_req_stagnation++;

    slave_request_stagnation = TRUE;
    }
else
    {
    turn = MASTER;
    }
}
/* a request from the slave queue was serviced */
slave_list_age = 0;
master_list_age++;
if ((master_list_age >= list_threshold) &&
        (!slave_request_stagnation))
{
if (q_head_is_not_empty(master_list))
    {
    num_master_list_stagnation++;

    master_list_stagnation = TRUE;
    turn = MASTER;
    }
}
```


EPNSDOCID: <EP 0676699A2 I >

```
                }
            else
                {
                /* no slave queue request was serviced */
                turn = MASTER;
                }
            }
        else
            {
            /* slave_list_stagnation or slave_request_stagnation */
            stagnant_operation = (resource_operation *)slave_list->head;
            master_list_age = 0;
            if (acquire_from_slave())
                {
                if (oldest_slave_serviced)
                    {
                    turn = MASTER;
                    slave_list_stagnation = FALSE;
                    slave_request_stagnation = FALSE;
                    master_list_age++;
                    slave_request_age = 0;
                    slave_list_age = 0;
                    }
                }
            else
                {
                if (acquire_from_master())
```


\SDOCID: <EP___0676699A2_I_>

```
                                {
                                master_list_age = 0;
                                if (oldest_master_serviced)
                                    master_request_age = 0;
                                }
                            }
                        }
                    }
                }

/*************************************************************************
***/
status acquire_from_master()
/*************************************************************************
***/
{
resource_operation *list_end, *operation, *first_operation;
node *operation_node;
int temp_channels_available;
int first_op_channels_available, other_op_channels_available;

oldest_master_serviced = FALSE;
if (q_head_is_not_empty(master_list))
    {
    list_end = (resource_operation *)master_list;
    first_operation = operation = (resource_operation *)master_list->head;
```


ENSDOCID. <EP 0676699A2 I >

```
first_op_channels_available = channels_available;
other_op_channels_available = channels_available;
if (master_list_stagnation || master_request_stagnation)
    {
    other_op_channels_available =
        channels_available ^ (channels_available & operation->channel_map);
    }
if (slave_list_stagnation || slave_request_stagnation)
    {
    first_op_channels_available =
    other_op_channels_available =
        channels_available ^ (channels_available & operation->channel_map);
    }

do
    {
    operation_node = (node *)operation;
    if (operation == first_operation)
        temp_channels_available = first_op_channels_available;
    else
        temp_channels_available = other_op_channels_available;

    if (operation->channel_map == (operation->channel_map &
            temp_channels_available))
        {
        /* channels are available for this operation; grant it */
        unlink_node(operation_node);
```


ISDOCID: <EP___0676699A2_I_>

```
                    link_q_tail(master_granted_list, operation_node);
                    channels_available &= operation->channel_map;
                    if (operation == first_operation)
                        oldest_master_serviced = TRUE;
                    last_serviced = MASTER;
                    age_request_age(master_list);
                    check_channel_use();

                    return(TRUE);
                    }
                operation = (resource_operation *)operation_node->next;
                }
            while (operation != list_end);
            return(FALSE);
            }
        else
            {
            return(FALSE);
            }
        }

        /*************************************************************************
        ***/
        status acquire_from_slave()
        /*************************************************************************
        ***/
        {
```


...SDOCID: <EP 0676699A2 I >

```
resource_operation *list_end, *operation, *first_operation;
node *operation_node;
int temp_channels_available;
int first_op_channels_available, other_op_channels_available;

oldest_slave_serviced = FALSE;
if (q_head_is_not_empty(slave_list))
    {
    list_end = (resource_operation *)slave_list;
    first_operation = operation = (resource_operation *)slave_list->head;
    first_op_channels_available = channels_available;
    other_op_channels_available = channels_available;
    if (slave_list_stagnation || slave_request_stagnation)
        {
        other_op_channels_available =
            channels_available ^ (channels_available & operation->channel_map);
        }
    if (master_list_stagnation || master_request_stagnation)
        {
        first_op_channels_available =
        other_op_channels_available =
            channels_available ^ (channels_available & operation->channel_map);
        }

    do
        {
        operation_node = (node *)operation;
```


ISDOCID: <EP___0676699A2_I_>

```
                if (operation == first_operation)
                    temp_channels_available = first_op_channels_available;
                else
                    temp_channels_available = other_op_channels_available;

                if (operation->channel_map == (operation->channel_map &
                    temp_channels_available))
                    {
                    /* channels are available for this operation; grant it */
                    unlink_node(operation_node);
                    link_q_tail(slave_granted_list, operation_node);
                    channels_available ^= operation->channel_map;
                    if (operation == first_operation)
                        oldest_slave_serviced = TRUE;
                    last_serviced = SLAVE;
                    age_request_age(slave_list);
                    check_channel_use();

                    return(TRUE);
                    }
                operation = (resource_operation *)operation_node->next;
                }
            while (operation != list_end);
            return(FALSE);
            }
        else
            {
```


ANSDOCID <EP 0676699A2 I >

```
        return(FALSE);
        }
    }                    /* end resource allocation algorithm */
```

Explanations and definitions for terms used in the above algorithm are provided below:

service_loops -- the number of requests that can be outstanding at any one time.

master_request_stagnation -- the state entered when the master ICON chip has serviced the slave icon requests too many times without servicing a master ICON's request. (Inter-ICON fairness parameter)

master_list_stagnation -- the state entered when a request on the master ICON's request list is 'aged' beyond a configurable threshold relative to other requests being serviced in the master ICON's request queue. (This is used to promote Intra-ICON list request fairness to ensure starvation within the master ICON's list is avoided because of a request requiring a large number of resources waiting behind many requests requiring only small numbers of resources.)

slave_request_stagnation -- the state entered when the master ICON chip has serviced the master ICON's requests too many times without servicing a slave ICON's request. (Inter-ICON fairness parameter)

slave_list_stagnation -- the state entered when a request on the slave ICON's request list is 'aged' beyond a configurable threshold relative to other requests being serviced in the slave ICON's request queue. (This is used to promote Intra-ICON list request fairness to ensure starvation within the slave ICON's request list is avoided because of a request requiring a large number of resources waiting behind lots of requests requiring only small numbers of resources.)

last_serviced -- a mechanism for providing fairness in servicing the least recently serviced controller.

turn -- indicates which list will be looked at first when servicing requests.

master_list_age -- the relative age of the request list for the master as compared to the number of requests serviced from the slave's list. It is used to ensure that the master is serviced at worst case, after some number of requests have been serviced from the slave. When the master list age exceeds a threshold, the master_request_stagnation state is entered into.

master_request_age -- the relative age of the oldest member of the master list when compared to the number of requests serviced from the master's list. It is used to ensure that the oldest request on the master's list is serviced at worst case, after some number of other requests have been serviced within the master list. When the master request age exceeds a threshold, the master_list_stagnation state is entered into.

slave_list_age -- the relative age of the request list for the slave as compared to the number of requests serviced from the master's list. It is used to ensure that the slave is serviced at worst case, after some number of requests have been serviced from the master. When the slave list age exceeds a threshold, the slave_request_stagnation state is entered into.

slave_request_age -- the relative age of the oldest member of the slave list when compared to the number of requests serviced from the slave's list. It is used to ensure that the oldest request on the slave's list is serviced at worst case, after some number of other requests have been serviced within the slave list. When the slave request age exceeds a threshold, the slave_list_stagnation state is entered into.

The algorithm presented above, together with the description of the invention provided earlier, should be readily understood by those skilled in the art as providing a method for managing the operations of multiple disk array controllers which share access to the disk drive units, busses, and other resources within the array.

Although the presently preferred embodiment of the invention has been described, it will be understood that various changes may be made within the scope of the appended claims.

Claims

1. A method of coordinating the execution of I/O requests received from requesting agents in a computer system in which said requesting agents share access to and control over a plurality of resources, characterized by the steps of:

   (A) establishing a request queu which includes an entry corresponding to each I/O requ st received from said requesting agents, each entry including an identification of resources that are required by said entry's corresponding I/O request;

   (B) maintaining a resources available status array which includes an entry for each resource which is

ISDOCID: <EP___0676699A2_I_>

not currently in use by any requesting agent and is not currently reserved for future use by any requesting agent;
(C) systematically comparing each entry in said request queue with the ntries in said resources available status array to detect an entry in said request queue identifying resources all of which are contained in said resources available status array;
(D) granting control of the resources associated with said entry detected in step C to the requesting agent providing the I/O request corresponding to the entry detected in step C; and
(E) executing the I/O request corresponding to the entry identified in step C.

2. A method as claimed in Claim 1 and further comprising the step of:
removing from said resources available status array the entries for each resource which is associated with said entry detected in step C upon the completion of step C.

3. A method as claimed in Claim 2 and further comprising the step of:
returning to said resources available status array the entries for each resource which is associated with said entry detected in step C upon the completion of step E.

4. A method as claimed in Claim 1, 2 or 3, wherein each entry in said request queue includes a request age indicating the relative age of each entry in the request queue with respect to other entries in the request queue, said method further including the step of:
granting priority to an entry in the request queue based on the relative ages.

5. A method as claimed in any one of the preceding claims wherein:
said resources include disk drives (31-35, 41-45) within a disk array; and
said requesting agents comprise disk array controllers (11,13) which share access to and control over said disk drives (31-35, 41-45).

6. A method as claimed in any one of Claims 1 to 4 wherein:
said requesting agents comprise disk array controller (11, 13) within a disk array subsystem within said computer system; and
said resources comprise disk drives (31-35, 41-45) and busses (51-55) within said disk array subsystem.

7. A method as claimed in any one of the preceding claims wherein:
said resources include busses (51-55) within said computer system.

8. A method as claimed in any one of Claims 1, 2 or 3 wherein:
each entry within said request queue further includes a request age indicating the relative age of each entry in said request queue with respect to other entries in the request queue; and
said method further includes the steps of:
examining said request ages to identify any entry having a request age which exceeds a predetermined request age value; and
removing from said resources available status array the entries for each resource which is associated with said entry having a request age which exceeds said predetermined request age value, reserving the resources for use with the requesting agent and request associated with the entry having a request age which exceeds said predetermined request age value.

9. A method as claimed in any one of the preceding claims wherein:
a request queue is maintained for each requesting agent in said computer system; and
said step of systematically comparing each entry in said request queue with the entries in said resources available status array includes the step of alternately examining the entries in each request queue to detect an entry in said request queues identifying resources all of which are contained in said resources availabl status array.

10. A method of coordinating the operation of processors, in a disk array system including first and second disk array controllers and a plurality of disk drives and busses under the control of each of said processors, characterized by the steps of:
(A) stablishing a request queue for each disk array controller which includes an entry corresponding

EPODOC ID: <EP 0676699A2 1 >

to an I/O requ st received by said disk array system, each entry including an identification of disk drives and busses that are required by said entry's corresponding I/O request;
(B) maintaining a resources available status array which includes an entry for each disk drive and bus which is not currently in use by any disk array controller and is not currently reserved for future use by any disk array controller;
(C) systematically comparing each entry in said request queues with the entries in said resources available status array to detect an entry in said request queues identifying resources all of which are contained in said resources available status array;
(D) granting control of the disk drives and busses associated with said entry detected in step C to the disk array controller associated with the request queue corresponding to the entry detected in step C; and
(E) executing the I/O request corresponding to the entry identified in step C.

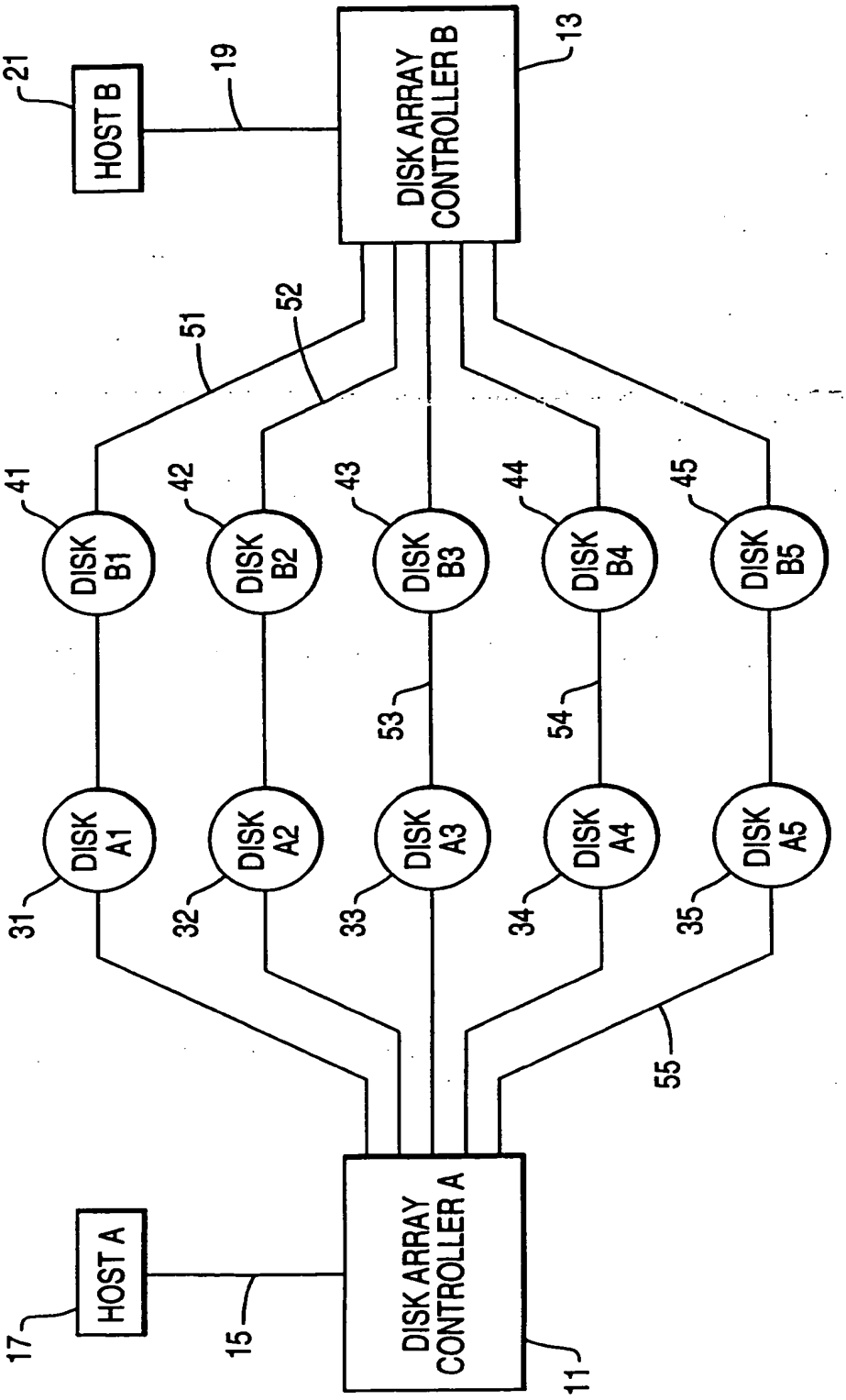
11. A method as claimed in Claim 10 wherein:
each entry within said request queue further includes a request age indicating the relative age of each entry in said request queue with respect to other entries in the request queue; and
said method further includes the steps of:
examining said request ages to identify any entry having a request age which exceeds a predetermined request age value; and
removing from said resources available status array the entries for each resource which is associated with said entry having a request age which exceeds said predetermined request age value, reserving the disk drives for use with the disk array controller and I/O request associated with the entry having a request age which exceeds said predetermined request age value.

12. A method as claimed in Claim 10 or 11, wherein:
said step of systematically comparing each entry in said request queue with the entries in said resources available status array includes the step of alternately examining the entries in each request queue to detect an entry in said request queues identifying resources all of which are contained in said resources available status array.

13. A method as claimed in Claim 10, wherein:
each request queue further includes a list age indicating the age of the oldest entry in the request queue;
said step of systematically comparing each entry in said request queue with the entries in said resources available status array includes the steps of:
examining said list ages to identify any request queue having a list age which exceeds a predetermined list age value; and
if a request queue is identified as having a list age which exceeds said predetermined list age, systematically comparing the entries in the request queue identified as having a list age which exceeds said predetermined list age only until the list age for the request queue is below said predetermined list age value.

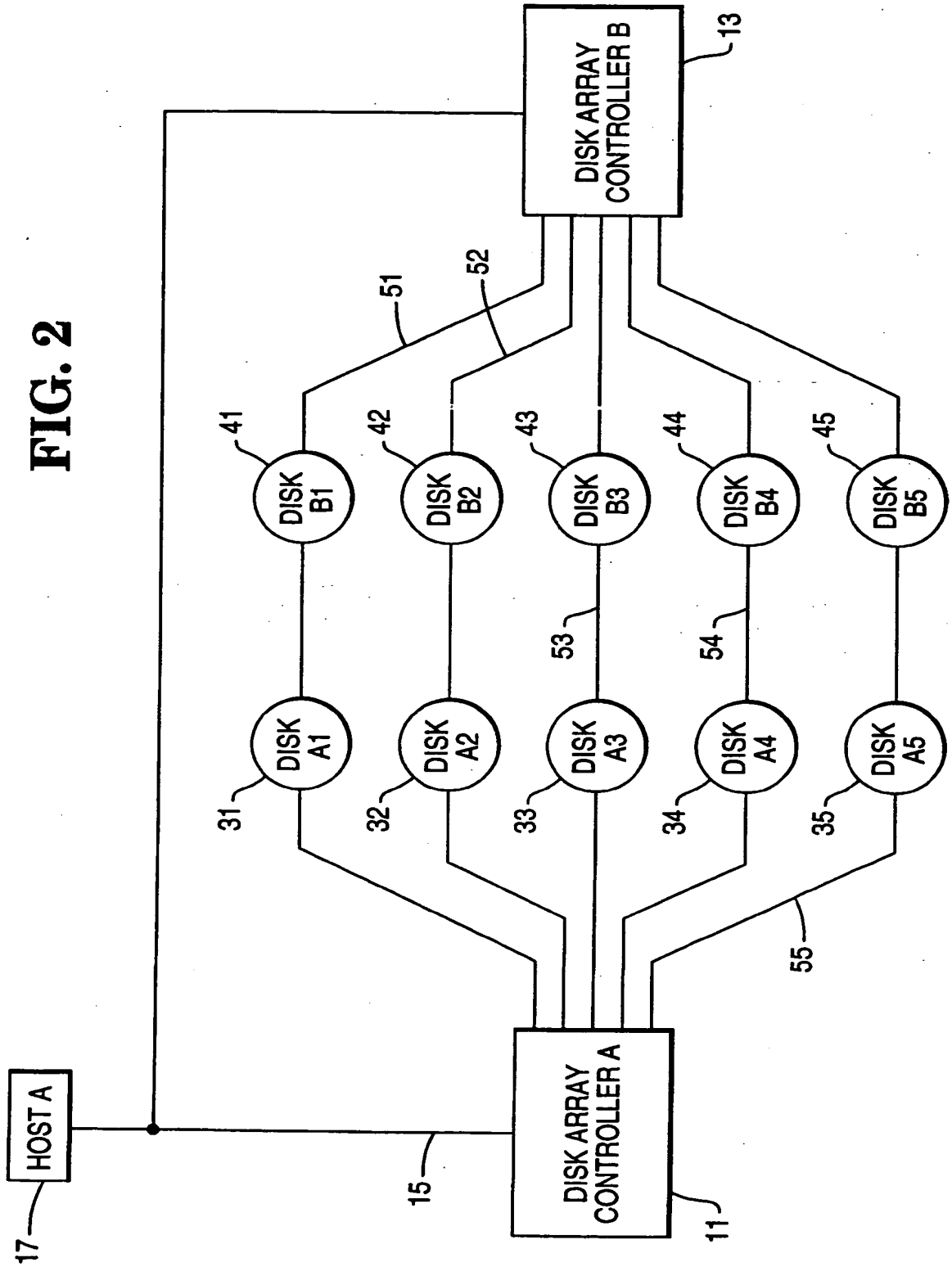
ISDOCID: <EP___0676699A2_I_>

FIG. 1



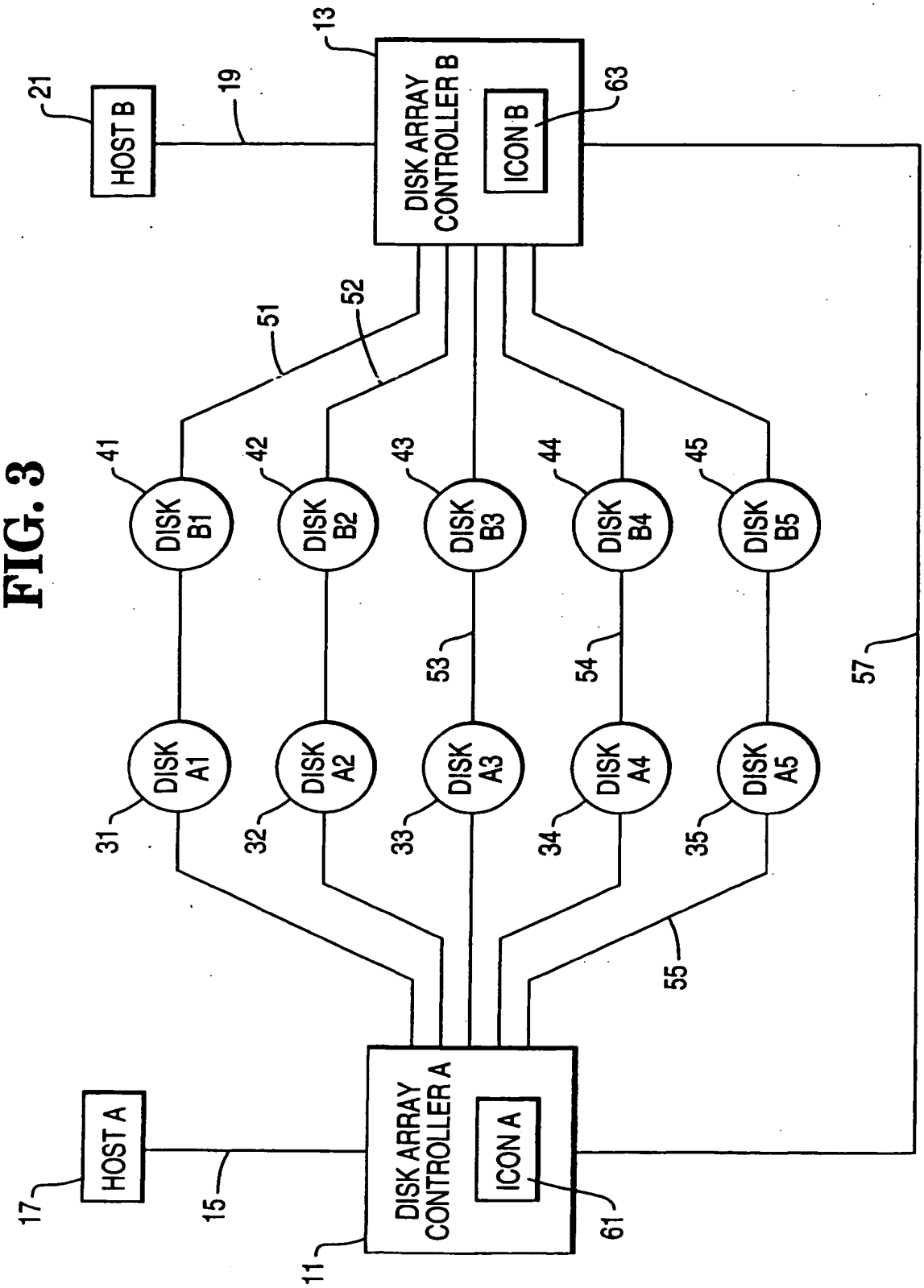
EPODOC: <EP 0676699A2 1 >

FIG. 2



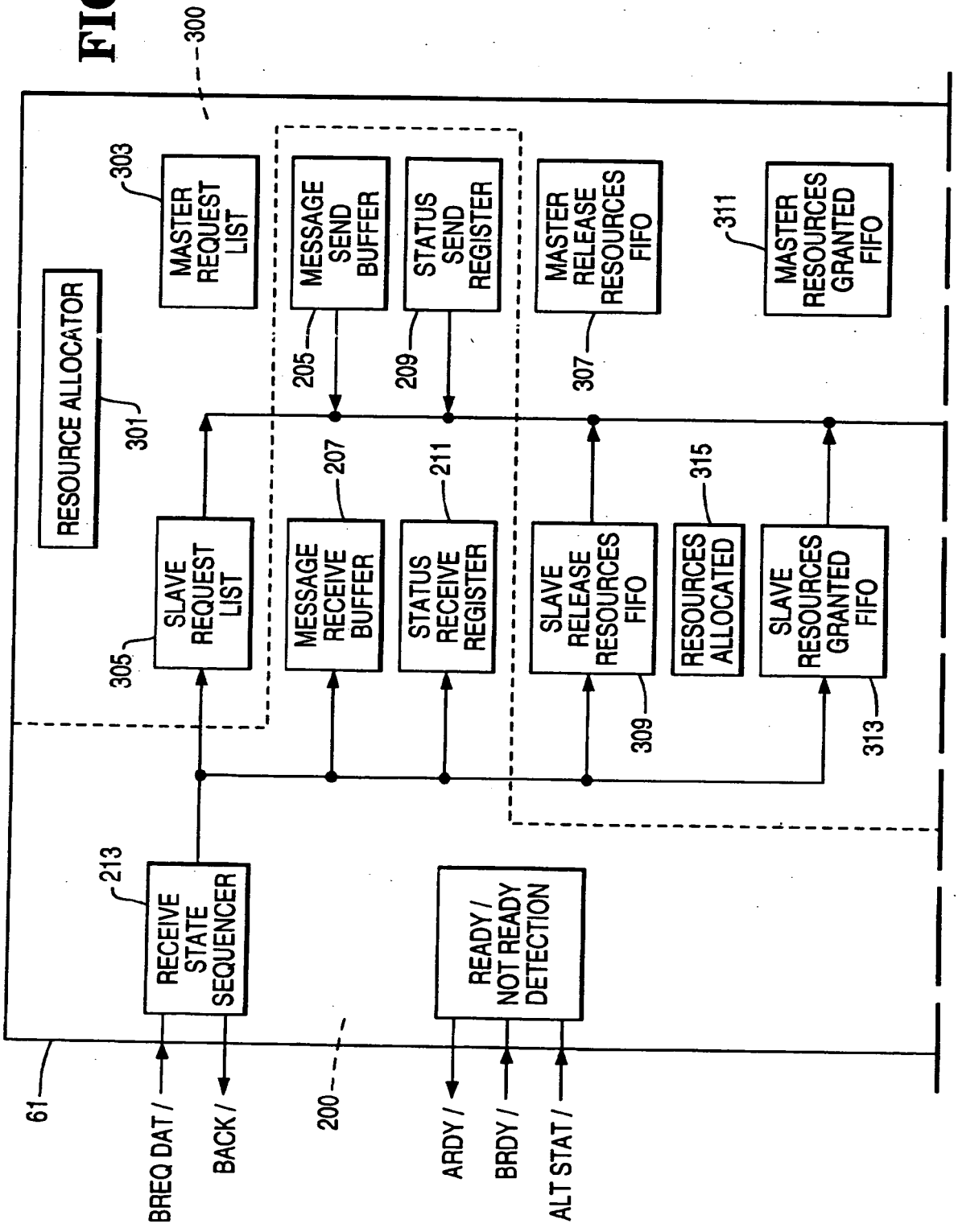
SDOCID: <EP___0676699A2_I_>

FIG. 3



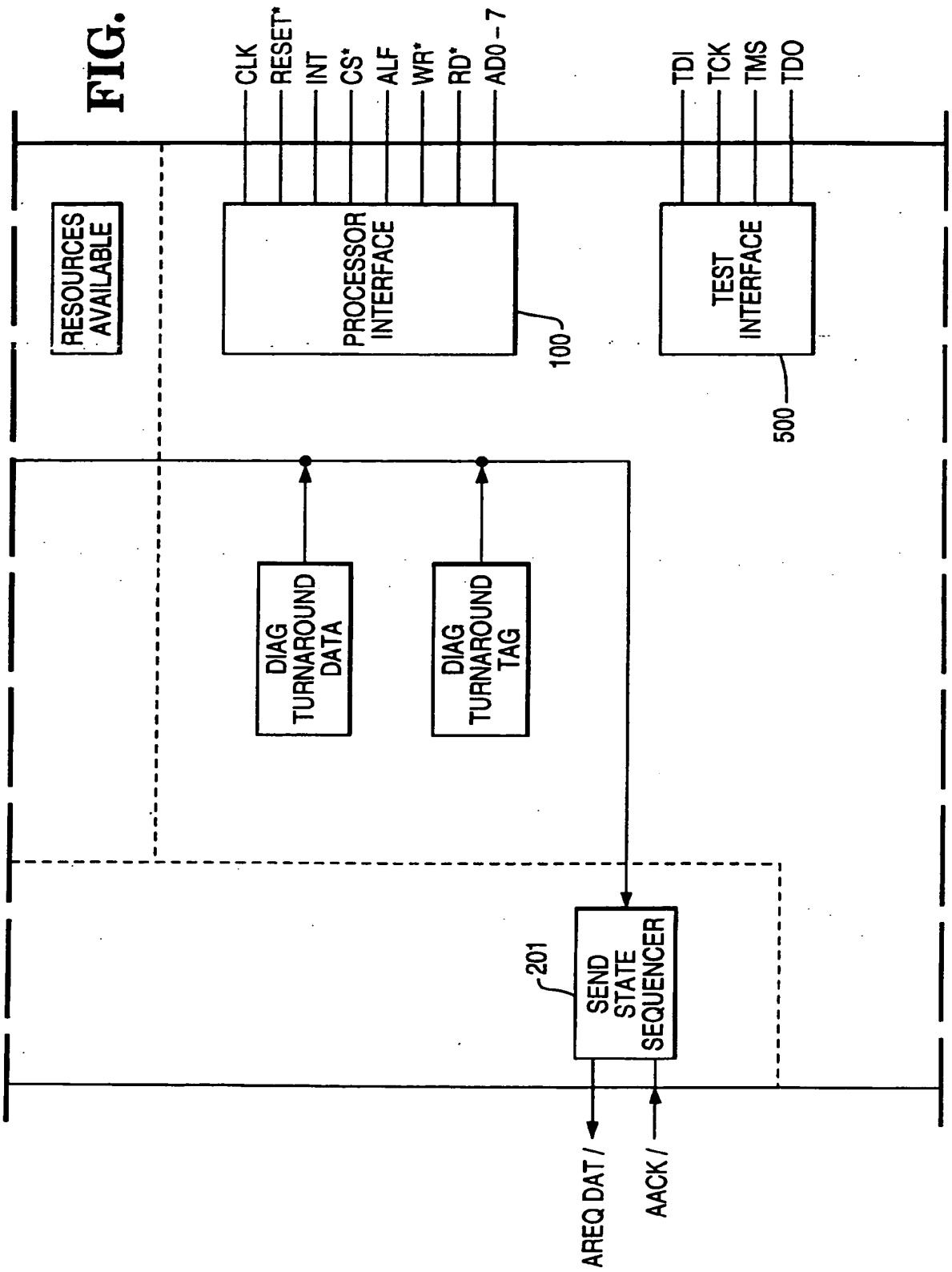
EPXCOID: <EP 0676699A2 I >

FIG. 4A



ISDOCID: <EP___0676699A2_I_>

FIG. 4B



...DOCID: <EP___0676699A2_I_>

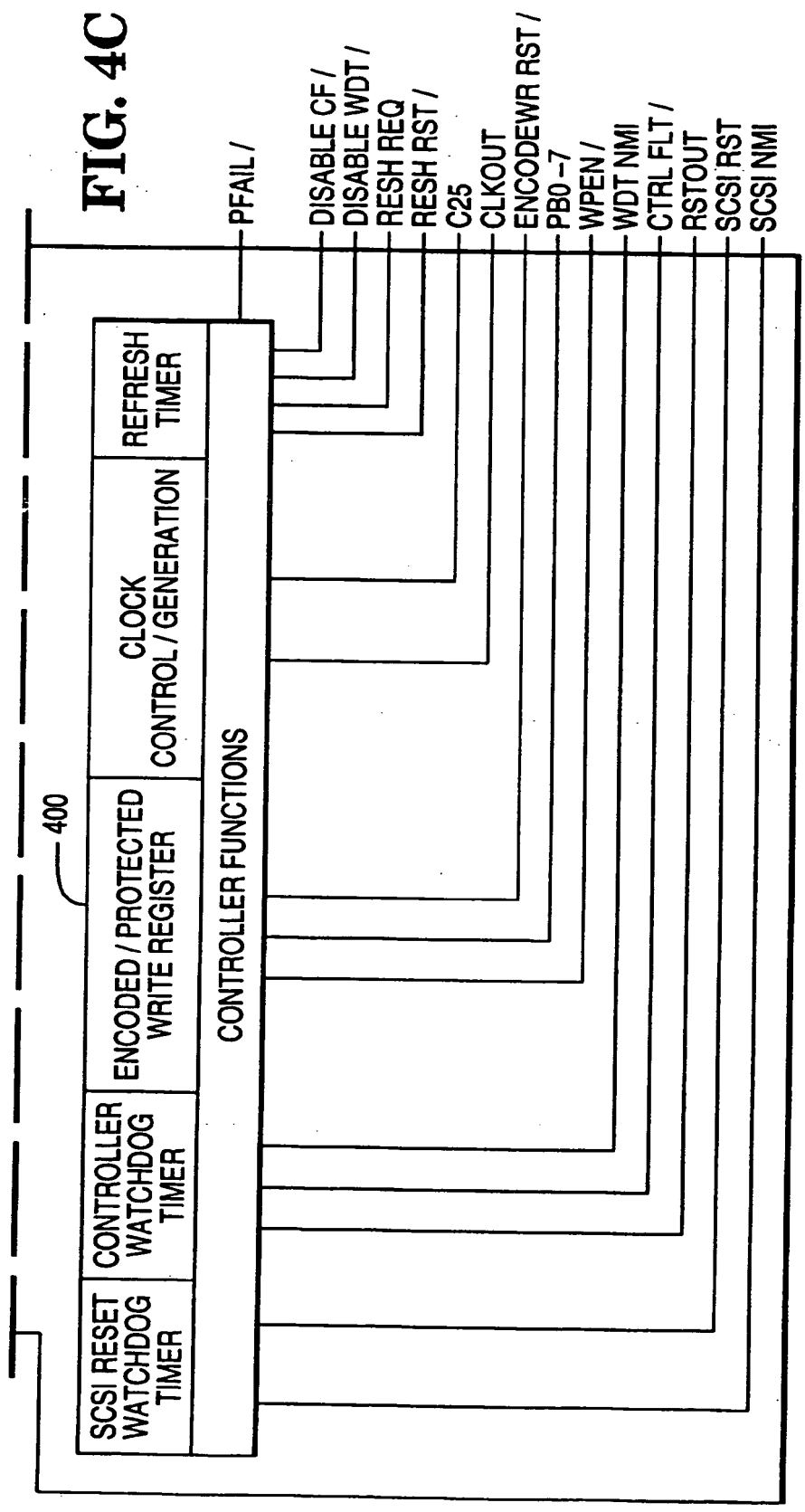


FIG. 4C

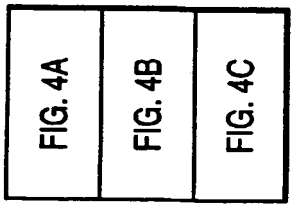


FIG. 4

ISDOCID: <EP___0676699A2_I_>

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 676 699 A3**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **95302002.1**

(51) Int. Cl.[6] : **G06F 13/18**

(22) Date of filing : **24.03.95**

(30) Priority : **04.04.94 US 223405**

(43) Date of publication of application :
11.10.95 Bulletin 95/41

(84) Designated Contracting States :
DE FR GB

(88) Date of deferred publication of search report :
18.10.95 Bulletin 95/42

(71) Applicant : **SYMBIOS LOGIC INC.**
2001 Danfield Court
Fort Collins, Colorado 80525 (US)

(72) Inventor : **Dekoning, Rodney A.**
6443 Danbury
Wichita, KS67226 (US)
Inventor : **Hoglund, Timothy E.**
3035 Purgatory Drive
Colorado Springs, CO 80918 (US)

(74) Representative : **Cill, David Alan**
W.H. Beck, Greener & Co.,
7 Stone Buildings,
Lincoln's Inn
London WC2A 3SZ (GB)

(54) Method of managing resources shared by multiple processing units.

(57) The invention provides for resource allocation logic for a computer system including a plurality of processors (17, 21) which share access to, and control of, a plurality of resources, such as disk drive units (31-35, 41-45) or busses (51-55). Resource allocation logic (300) coordinates the execution of requests received from the processors to avoid resource sharing inefficiencies and deadlock situations. The allocation logic (300) maintains a "request" queue for each processor (17, 21), seeking to satisfy all requests quickly and fairly. The queues contain an entry corresponding to each request received from its corresponding processor and an identification of resources that are required by the entry's corresponding request. The allocation logic (300) also maintains a "resources available" status array of resources which are not currently in use by any processors, or are not reserved for future use by any processors. The logic repeatedly compares each entry in the request queues with the entries in the resources available status array to detect an entry in the request queue identifying resources all of which are contained in the resources available status array. Once the allocation logic (300) can satisfy a particular request, it signals a grant to the requesting processor for the resources requested and the requested resources are removed from the resources available status array. Upon conclusion of execution of the granted request, the resources are again released to the resource allocation logic (300) for utilization by other resource requests.

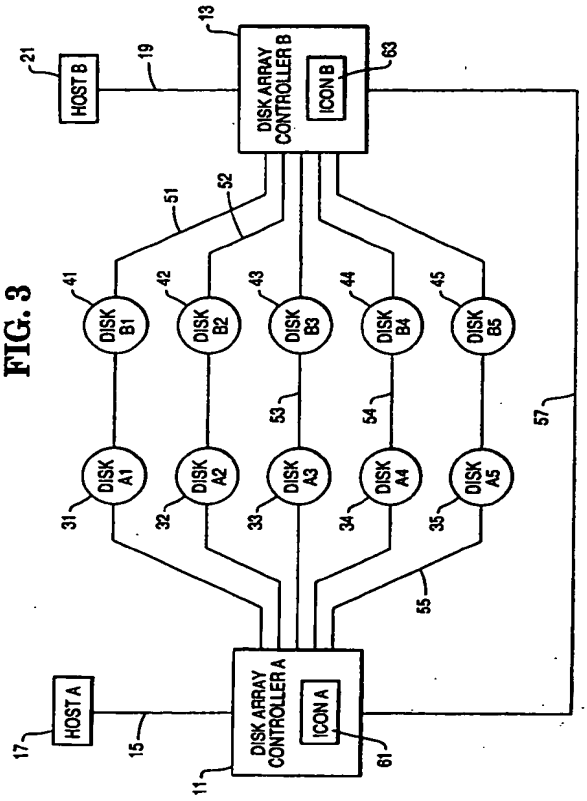


FIG. 3

EP 0 676 699 A3

SNSDOCID: <EP___0676699A3_I_>

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 30 2002

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
Y	GB-A-2 017 363 (HONEYWELL INFORMATION SYSTEMS INC.) * page 1, line 72 - page 2, line 3 * * page 2, line 18 - line 76 * * page 3, line 58 - page 4, line 15 * * figure 1 *	1-9	G06F13/18
Y	EP-A-0 476 252 (INTERNATIONAL BUSINESS MACHINES CORP.) * column 1, line 46 - column 2, line 34 * * column 5, line 4 - column 6, line 27 * * column 7, line 19 - line 42 * * claims 1-7 *	1-4,8,9	
Y	EP-A-0 426 184 (COMPAQ COMPUTER CORP.) * page 4, line 48 - page 5, line 15 * * page 7, line 55 - page 8, line 52 * * page 15, line 1 - line 28 * * claims 1-6 *	5-7,10	
Y	PATENT ABSTRACTS OF JAPAN vol. 18 no. 327 (P-1757) ,21 June 1994 & JP-A-06 075843 (NEC SOFTWARE LTD.) 18 March 1994, * abstract *	10	TECHNICAL FIELDS SEARCHED (Int.Cl.6) G06F
A	EP-A-0 298 793 (SERVEL ET AL) * column 1, line 51 - column 3, line 7 *	1-13	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	24 August 1995	McDonagh, F

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

SDOCID: <EP___0676699A3_I_>